JSAGA: using SAGA for uniform access to heterogeneous grid infrastructures

OGF 23, Barcelona
Sylvain Reynaud

http://grid.in2p3.fr/jsaga/
Plan

- Overview of **JSAGA implementation**
  - SAGA implementation
  - Additional API

- Feedback on **SAGA specification**
  - Successes
  - Compatibility-breaks required to enable our use cases
About CC-IN2P3

- High throughput data processing facility
  - not co-located with an experimental site
- Mission: 24x7 service
  - mass storage repository
  - high-throughput computing facilities
  - technical consulting and training services for laboratories
  - web hosting, video-conferencing, e-mail infrastructure, news, wiki, webcast, …

- Users
  - ≈2000 local accounts + grid users
  - 70 research groups, mainly international collaborations in nuclear physics, particle physics and astro-particle physics
  - also bio-medical apps.
Involved since 2001 in grid projects at international, national and regional levels

- EU Datagrid, EGEE, WLCG, EELA-2
- OpenPlast
  - simulation of polymer injection
- Rugbi
  - Grid-based platform for bio-informatics applications
- GridPPS
  - protein pattern scanning

OpenPlast & Rugbi projects were lead by industry partners
Motivation: uniform access to grids

different grid characteristics
• availability of data
• large number of computing resources
• specific needs (e.g. super-computer, confidentiality)
• small overhead (e.g. consolidation of results)
What is JSAGA?... an example
What is JSAGA?... an example
What is JSAGA?...

Both implementer and user of the SAGA specification

A Java implementation of the SAGA specification for:
- hiding heterogeneity of middleware

A higher-level API for:
- efficiently submitting job collections
- hiding heterogeneity of grid infrastructures
- Simple API for Grid Application
  - object-oriented API
  - to easily develop grid applications
    - << functional packages for fundamental programming capabilities >>

- Plug-ins API
  - service-oriented API
  - to easily develop plug-ins that enable the engine to efficiently provide SAGA features
    - by implementing low-level methods
    - by exposing supported methods & techno
    - by throwing homogeneous exceptions
Data and execution plug-ins
- declare
  - supported security plug-ins

Security plug-ins
- declare
  - attributes usage rules
  - attributes default values
- engine set the context instance to first matching usage rule

example: usage rules for a globus proxy
/etc/grid-security/certificates
/tmp/x509up_u_$UID
~/.globus/user*.pem

- supported security plug-ins
  - attributes usage rules
  - attributes default values
- engine set the context instance to first matching usage rule

example: usage rules for a globus proxy
Data
The engine combines the available plug-in methods to efficiently provide the functionality requested by the user
- e.g. third-party transfer

The engine enables caching
- opened connections
- information about entries
- content of entries through
Execution

Execution

Data

Security

Execution

Job control

Job monitoring
Monitoring and job control are provided by separate plug-ins
- e.g. monitor with gatekeeper, gLite-LB or gsiftp LIST command

Monitoring strategy of the engine depends on
- monitoring capabilities of the plug-in
- monitoring activities of the user
  - last and other on-going activities
Execution

- Streaming job standard input/output/error may be done...
  - before/after/during job execution
  - through direct/buffered/piped stream

- Streaming strategy of the engine depends on
  - streaming capabilities of the plug-in
  - streaming activities of the user
A Java implementation of the SAGA specification for:
• hiding heterogeneity of middleware (e.g. gLite, Globus, Unicore)

A higher-level API for:
• efficiently submitting job collections
• hiding heterogeneity of grid infrastructures (e.g. EGEE, OSG, DEISA)
Execution of collection + Staging

commands

jsaga-context-init
jsaga-context-info
jsaga-context-destroy
jsaga-cat
jsaga-cp
jsaga-ls
jsaga-logical
jsaga-mkdir
jsaga-mv
jsaga-rm
jsaga-rmdir
jsaga-test
jsaga-job-run
jsaga-job-submit
jsaga-job-status
jsaga-job-getoutput
jsaga-jobcollection-run
jsaga-jobcollection-submit
jsaga-jobcollection-status
jsaga-jobcollection-getoutput

interfaces

SAGA API

Context
File
Job

JSAGA API

Job Collection

implementation

JSAGA ENGINE

Security

Data

Execution

Exec. + Staging

control
monitor

protocol

language
expression

additional API
Execution of collection + Staging

Exec. + Staging

Security

Data

Execution

Exec. + Staging
Execution of collection + Staging

Plug-ins translate job description to...

- Job Collection
  - parametric
  - SDL

- Job
  - JSDL
    - RRL
    - JLML

- Data Staging extensions:
  - staging alternatives,
  - pre/post-processing,
  - sandbox flag...

- termination rules
  - (temporal + status)

- retry rules
  - (temporal + status)

Language specific

JSDL + ext.
Execution of collection + Staging

- **basic**
  - Get parametric job index

- **default**
  - Get system properties (-Dxxx)
  - Evaluate math expressions
  - Manipulate strings
  - Format dates and floats

- **JEP**
  - Java Math Expression Parser
  - [https://sourceforge.net/projects/jep/](https://sourceforge.net/projects/jep/)

- **Bean Shell**
  - Lightweight Scripting for Java

```xml
... <POSIXApplication>
    <Executable>myexec</Executable>
    <Argument>from @{date('dd/MM/yyyy', now+24*INDEX)}</Argument>
... </POSIXApplication>
```
Close to SAGA look & feel

Session session = SessionFactory.createSession();
JobCollectionManager manager = JobCollectionFactory.createJobCollectionManager(session);
JobCollectionDescription description = JobCollectionFactory.createJobCollectionDescription("J SDL", descriptionStream);
TaskContainer collec = manager.createJobCollection(description);
collec.run();
((JobCollection) collec).bindResourceManagers(new URL[]{...});
while (jc.size() > 0) {
    Task job = collec.waitFor(WaitMode.ALL);
    System.out.println(job.getState());
}
Description of used infrastructures

- Hide middleware heterogeneity
  - Uniform interface (SAGA)

- Hide infrastructure heterogeneity
  - Selection of the right security context
  - Selection of the right client API configuration
  - Transport of the job input/output data to/from worker nodes

→ Describe the capabilities of the infrastructures you want to use
Description of used infrastructures

- Hide middleware heterogeneity
  - Job management technologies
    - e.g. CREAM, WMS, SSH, GK

- Hide infrastructure heterogeneity
  - Computing Elements (e.g. GK)
    - Different grid or site policies
      - e.g. network filtering, shared FS
    - Different environment variables
      - e.g. $VO_?_SW_DIR, /usr/local
    - Different configuration attributes
      - e.g. monitor service URL, shell path on cygwin, default SE URL
    - Commands available on worker
      - e.g. globus-url-copy, srmcp, Scp, wget, tar
Transfer path depends on...

- Using a single grid:
  - All files can be transferred to the worker through a single node.

- Using several grids:
  - Need to dynamically build a transfer graph, according to...

Additional API:

- WMS
- VOMS
- Globus
- gatekeeper
- srb://
- srm://
- ifn://
- gsfip:// (v1)
- gsfip:// (v2)
- http://
- tar://
- localhost
- World
- Grid
- OpenPlast
- EGEE

JSAGA
Transfer path depends on...

- grid or site
  - network filtering policy
  - commands available on workers
  - services available from workers (close Storage Element, shared FS)
  - supported context instances

- execution service
  - protocols supported for staging

- transfer protocol
  - access mode (RO, WO, RW)
  - third-party transfer
  - supported data protection level

- data to stage
  - shared by several jobs
  - installed on some worker nodes
  - file size
  - required data protection level

Additional API

JSAGA
Transfer path depends on...

- grid or site
  - network filtering policy
  - commands available on workers
  - services available from workers (close Storage Element, shared FS)
  - supported context instances

- execution service
  - protocols supported for staging

- transfer protocol
  - access mode (RO, WO, RW)
  - third-party transfer
  - supported data protection level

- data to stage
  - shared by several jobs
  - installed on some worker nodes
  - file size
  - required data protection level
Transfer path depends on...

- grid or site
  - network filtering policy
  - commands available on workers
  - services available from workers (close Storage Element, shared FS)
  - supported context instances

- execution service
  - protocols supported for staging

- transfer protocol
  - access mode (RO, WO, RW)
  - third-party transfer
  - supported data protection level

- data to stage
  - shared by several jobs
  - installed on some worker nodes
  - file size
  - required data protection level

- execution service
  - protocols supported for staging

- transfer protocol
  - access mode (RO, WO, RW)
  - third-party transfer
  - supported data protection level

- data to stage
  - shared by several jobs
  - installed on some worker nodes
  - file size
  - required data protection level
Example of generated graph

Data flow

Several protocols used, but only 3 jobs submitted on 1 grid…
Summary: layered interfaces

- **For end-users**: ready-to-use software adapted to their needs
- **For application developers**: hide infrastructure heterogeneity
- **For application developers**: hide middleware heterogeneity
- **For plug-in developers**: As many interfaces as ways to implement the functionalities

Many plug-ins contributed by
Related projects

- JSAGA is being integrated into…
  - Elis@ • a web portal for submitting jobs to industrial and research grid infrastructures
  - JJS (Java Job Submission) • a tool for submitting job collections to EGEE
    • optimized for short-life jobs (resource selection based on QoS observed while submitting jobs)
  - JUX (Java Universal eXplorer) • a multi-protocol file browser
Feedback on SAGA
Feedback on SAGA specification

SAGA suits most of our needs, it is a sound foundation
  - for hiding middleware heterogeneity
  - with a consistent specification

→ With our hindsight, we would do this choice again if we had to start from scratch today.

However, we had to break SAGA compliance to fulfill some of our needs....
Feedback on SAGA specification

- **Broken SAGA compliance for data management**

  - **Flag**
    - BYPASSEXIST added
      - subsequent methods may throw an IncorrectState exception

  - **Method**
    - copyFrom() added
      - to improve efficiency of file copy from many-to-1 server without caching connections

- **For optimization**

- **Methods**
  - remove() allow no RECURSIVE flag for directories
    - enable rmdir-like behavior
  - getLastModified() added
    - needed to implement a "ls -l" command
Feedback on SAGA specification

- **Broken SAGA compliance for execution management**

  - **Job description attributes**
    - JobStartTime and JobContact not supported
      - not supported by JSDL
      - not needed by our users
    - JobName added
      - needed for job collection management
    - CPUArchitecture and OperatingSystemType defined as scalar attributes
      - for consistence with JSDL

  - **metric "job.sub_state"**
    - "job.state" not detailed enough
      - RUNNING: QUEUED/ACTIVE
      - STAGING: INPUT/OUTPUT
    - "job.state_detail" not usable (different for each middleware)

  - **method bindResourceManager**
    - late binding of the job to a resource manager
    - third-party resource selection
      - SAGA job description contains no resource requirement info.
Feedback on SAGA specification

- SAGA job description
  - easy to use
  - suitable for hiding **middleware heterogeneity**
  - used by...

- JSDL
  - structured
  - can be extended for hiding **grid infrastructure heterogeneity**
    - e.g. data staging alternatives, pre/post-processing, sandbox flag...
  - used by...

---

Job Collection

SAGA
Feedback on SAGA specification

- We do not plan to implement the following **Core API** areas
  - streaming
  - remote procedure call

- because
  - not usable for uniform access (many middleware do not support them)
  - not required by our users

  ⇒ JSAGA is "partially SAGA compliant"

- We plan to implement the following **Extensions** area
  - service discovery
Feedback on SAGA java binding

- JSAGA implements the reference java binding of SAGA
  - :pserver:cvs_anon@cvs.cct.lsu.edu:/projects/SAGA-RG
  - currently implemented version is "snapshot-1"
- fully satisfied with it…

- …except for class URL: need to patch it to support
  - windows paths (file://c:/windows, file://C:\windows)
  - relative paths (file://./dir/file.txt)
  - special characters in paths (%\#)
  - etc.

/**
 * URL class as specified by SAGA. The java.net.URL class is not usable because
 * of all kinds of side-effects. TODO: provide factory with this as a default
 * implementation???
 */

public class URL {
  private URI u;

this would enable us avoid patching the java binding!
Layered interfaces

- For end-users: ready-to-use software adapted to their needs
- For application developers: hide infrastructure heterogeneity
- For application developers: hide middleware heterogeneity
- For plug-in developers: As many interfaces as ways to implement the functionalities
SAGA interface

- close to application developer needs
  - object-oriented
  - uniform interface to all the supported technologies
- not designed for being used as an interface for plug-ins
  - code for JSAGA engine is twice bigger than for all the plug-ins
Plug-ins interface

- close to existing middleware API
  - service-oriented
  - as many interfaces as ways to provide the functionalities
Plug-ins interface

- close to existing middleware API
  - service-oriented
  - as many interfaces as ways to provide the functionalities
- not designed for being used by application developers
Plug-ins interface

- Many grid tools (e.g. SAGA implementations, meta-schedulers, grid portals, workflow engines…) define their own interface for plug-ins
  - drawback: we have to develop specific plug-ins for each tool…
  - proposal: a specification for plug-ins would enable factorizing efforts
    - not only for SAGA implementations but also other grid tools

- close to existing middleware API
  - service-oriented
  - as many interfaces as ways to provide the functionalities

- not designed for being used by application developers
Conclusion

- **SAGA successfully** matches
  - the needs of our early adopter users
  - our needs for developing a job collection manager, which hides the heterogeneity of the grid infrastructures

- but we needed to add some methods, flags and attributes to fulfill some of our needs
  - this does not break SAGA-compliance of the **JSAGA engine**
  - but it breaks SAGA-compliance of the **user code** if he uses these features…

- a **specification for plug-ins** would enable factorizing the efforts of wrapping middleware API to uniform interfaces
Backup slides
- Minimize dependencies
  - resolve conflicts between versions of libraries
    - with maven
  - regenerate stubs from WSDL
    - with Axis 1.4
  - remove external dependencies
    - no gLite-UI for EGEE, no OS-specific dependency

- Provide an installer
  - install only what you need...